The Report Committee for Vignesh Radhakrishnan
Certifies that this is the approved version of the following report:

Bit Error Tolerance of Deep Neural Network Accelerators

APPROVED BY

SUPERVISING COMMITTEE:

Nur A. Touba, Supervisor

Mark McDermott

Bit Error Tolerance of Deep Neural Network Accelerators

by

Vignesh Radhakrishnan

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2020

Bit Error Tolerance of Deep Neural Network Accelerators

Vignesh Radhakrishnan, M.S.E.
The University of Texas at Austin, 2020

Supervisor: Nur A. Touba

The resurgence of machine learning in various applications and it's inherent compute-intensive nature require hardware accelerators in the edge devices. The underlying process technology is prone to faults. Hence, there is a need to make these hardware accelerators dependable. Deep Convolutional Neural Networks perform well for machine learning applications like image classification. This report presents the impact of bit errors on the DNN's performance. Most accelerators are designed with a one data type that fits all approach. The sensitivity of the DNNs with a single-precision floating-point format is studied. Due to the high sensitivity of the deep layers to critical bit errors and rapid performance degradation with increasing BER, several potential techniques to actively improve fault tolerance are discussed.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep Neural Networks (DNN) have increasingly been gaining interest because of their great performance on the task of image classification. Hence, there is a need for deployment of these networks on the Internet of Everything (IoE) Edge devices. Several hardware DNN accelerators have been proposed [1] [2]. Training is usually done offline and the trained parameters are loaded on to these hardware accelerators for real-time inference. However, DNNs have several convolutional layers with thousands of parameters per layer. For example, the last convolutional layer of AlexNet [9] Architecture has about 885K parameters, while the complete network has about 62M parameters. This obviously, consumes a large memory footprint and requires off-chip DRAM accesses that can be expensive in terms of energy consumption. Recent works employ techniques to efficiently use on-chip SRAMs and embedded DRAMs [1]. Eyeriss (shown in figure 1.1) is a DNN accelerator with 108KB of on-chip SRAM. Pre-trained weights can be moved from off-chip DRAM and a novel

Figure 1.1 is adapted from "Y. Chen, T. Krishna, J. Emer, and V. Sze. 14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks.In 2016 IEEE International Solid-State Circuits Conference (ISSCC),pages 262–263, 2016"

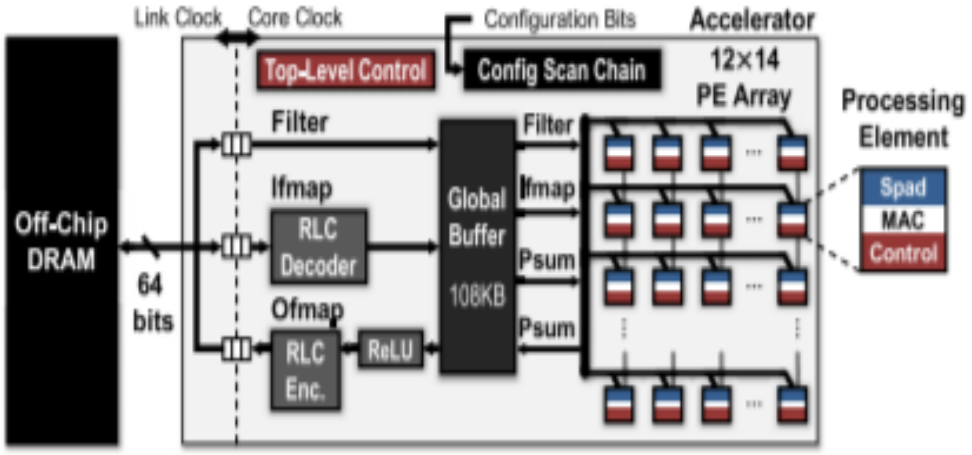

Figure 1.1: Eyeriss System Architecture [1]

data movement technique efficiently utilizes the on-chip buffer for inference.

Nevertheless, the parameters stored in the memory are crucial for the correct functioning of these networks. As process technology scales, a major part of the chip area can be consumed by the memory. Hence, the faults and errors in the memory can affect the reliability of the neural networks run on DNN accelerators. As a result, studying the sensitivity of DNNs to bit errors is important.

This report analyzes the performance impact of some popular DNNs due to errors and provides a survey of techniques that can make them fault-tolerant. Chapter 2 presents the background on fault models and DNNs used in the analysis. In Chapter 3, the experimental setup for error injection and results on performance impact is discussed. Chapter 4 provides a survey of

fault-tolerant techniques in the literature to improve the performance of the DNN accelerators. Finally, Chapter 5 summarizes and concludes the report.

Chapter 2

Fault Modeling of Neural Networks

The three fundamental components: fault, error and failure have a cause-effect relationship (Figure 2.1) from a physical standpoint to the behavior of the system. Faults are the physical defects/conditions in the system design that can give rise to errors. Errors are the deviation of the logical state or output of the system from the expected values. While faults can cause errors, failures that restrict the system's ability to perform intended operation may or may not occur. This essentially means that faults can be active causing errors and failures or might be deactivated.

For ease of fault detection and correction, researchers develop fault models that can describe the physical manifestation of faults and how they will affect the behavior as accurately as possible. The two widely used fault models are:

Figures 2.1 and 2.2 adapted from "C. Torres-Huitzil and B. Girau.Fault and error tolerance in neural networks: A review.IEEE Access, 5:17322–17341, 2017". Figure 2.3 is adapted from "K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016".

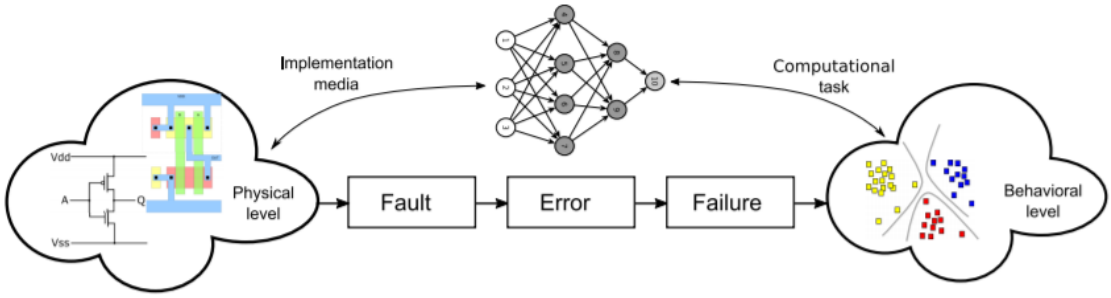

Figure 2.1: Cause-effect relationship between fault, error and failure [13]

- **Stuck-at model** where the data or a line appears to be held at logic high (stuck-at-1) or low (stuck-at-0)
- **Random bit flips**, a memory element has incorrect random value.

Generally, fault tolerance can be classified into passive and active fault tolerance. While active fault tolerance requires specialized structures that detect and correct faults, passive fault tolerance is the ability of the system to ensure correct outputs despite the faults. Neural networks are usually regarded as fault-tolerant (passively) because of the inherent redundancy built into it by overprovisioning of weights. However, the relationship between the intrinsic potential to tolerate faults and the actual number of faults have not been fully investigated.

A neural network N performing a computation H_N is said to be fault-tolerant [13] if the computation H_{Nfault} , performed by a faulty network N_{fault} obtained from N , is close to H_N. Formally, for $\varepsilon > 0$, N is called ε - fault-

tolerant, if it tolerates faulty components for any subset of size at most n_{fails}:

$$|| \; H_N(X) \; - \; H_{Nfault}(X) \; || \; \leq \; \varepsilon \; \; \forall \; X \in T$$

where X is the input stimulus part of the dataset T.

In the neural network (abstract model shown in Figure 2.2), an error can occur in the neuron in one of the following ways:

- An unexpected value in the input due to faulty interconnection or noise.
- In the weight due to bit errors in the memory.
- In the compute logic that performs summation or activation.

While stuck-at-faults can be used to model the first and last types of fault, random bit flips can be used to model the bit errors in the memory. These errors happen in memory element because of external perturbations, like single event upset. This report analyzes the tolerance and sensitivity of neural networks to random bit flips that can happen over the trained parameters stored in the memory.

In this report, the following two deep neural networks are chosen to be studied:

ResNet: Convolutional feedforward network that connects the output of the m^{th}layer to the input of the $(m+1)^{th}$ layer. ResNets also add skip-connection that bypasses the non-linear activation (Figure 2.3) function with

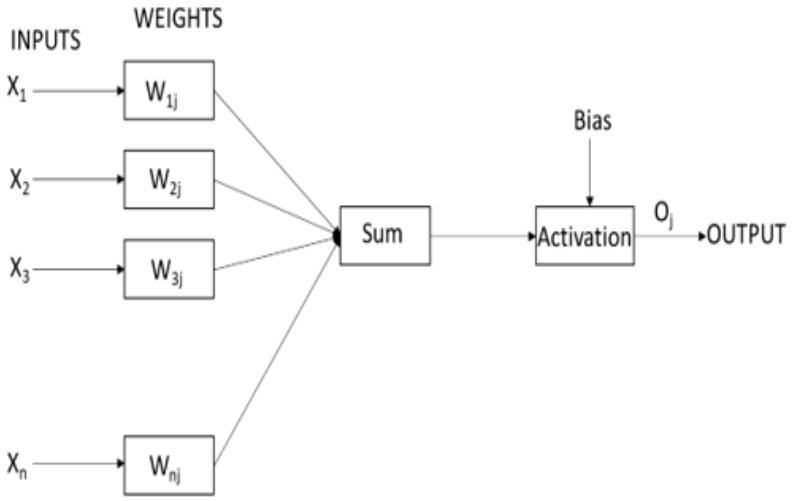

Figure 2.2: Abstract Model of a Neuron

an identity function:

$$x_m \;=\; H_m(x_{m-1}) \;+\; x_{m-1}$$

An advantage of ResNets is that gradient flows directly from later layers to the earlier layers through the identity function. In this report, two versions of ResNet [5]: ResNet20 and ResNet32 are analyzed.

The first layer of both the versions of ResNet is a 3x3 convolutional layer followed by a total of 6n layers. After the final convolution, a global average pooling layer is introduced after which the features are flattened and given to a k-way fully connected layer where k is the number of classes in the problem. Furthermore, each convolutional layer is followed by a batch normalization layer and ReLu activation. Note that the batch normalization layer is simply a shift and scale layer during inference. The total number of

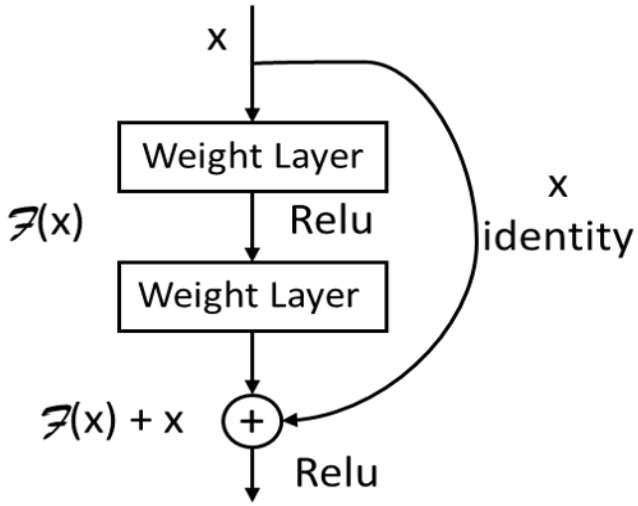

Figure 2.3: Residual Learning - Building Block

trainable parameters in the ResNet is shown in Table 3.1.

DenseNet: DenseNet [6] is very similar to ResNet with dense connectivity, every layer is connected to every other layer in a feedforward fashion. Consequently, the m^{th} layer receives feature maps of all preceding layers:

$$x_m \ = \ H_m(x_0, x_1, \ x_2, \ ... \ x_{m-1})$$

The hyperparameter k, growth rate, controls the number of input feature maps for a m^{th} layer as $k \times (m-1) + k_0$, where k_0 is the number of channels at the input. This growth rate prevents the network from becoming too wide. Thus, it improves parameter efficiency as compared to other deep networks. In this report, we use the DenseNet BC-100 configuration which has a growth rate of 12 and 100 convolutional layers.

Chapter 3

Experimental Setup and Results

The DNNs mentioned in Chapter 2, were trained for an image classification problem on CIFAR-10 Dataset [8]. The CIFAR-10 dataset consists of 50,000 training images and 10,000 testing images in 10 classes. All the images are colored and have a natural scene with 32 x 32 pixels each. Although a classification problem with just 10 classes might seem simple, CIFAR-10 is widely used dataset which was benchmarked by several DNNs. Hence, this dataset was chosen as opposed MNIST digits recognition dataset which is a similar classification problem. Besides, a very interesting relationship between overprovisioning and fault tolerance needs to be studied. Therefore, CIFAR-10 would be a better problem than ImageNet for these DNNs. All the images were preprocessed by normalization with their mean and standard deviation. The experiments were done on the Keras platform using the TensorFlow backend.

The following DNNs were trained using the training image set: ResNet 20, ResNet 32 and DenseNet for 200 epochs. In each epoch, the trained network was validated with the testing images and the iteration which gave the highest validation accuracy was picked. Table 3.1 shown below gives information about the training setup and achieved validation accuracy on each of the

networks.

Table 3.1: Results of DNNs trained on CIFAR-10

Network	**# of Parameters**	**Optimizer**	**Validation Accuracy**
ResNet20	0.27	ADAM	0.9161
ResNet32	0.47	ADAM	0.9216
DenseNet	0.79	RMSProp	0.9368

The fault injection experiments were done in the feedforward path during the inference stage. Note that the data type of the weights and MAC computation done in these experiments are 32-bit single-precision floating-point. The following procedure was adopted for error injection:

1. Choose a Bit Error Rate (BER).
2. Calculate the total number of bits (n_{total}) in the pre-trained weights for the entire network.
3. Calculate the total number of bit errors to be injected $n_\epsilon = (BER) \times (n_{total})$.
4. Repeat steps 5,6,7 for n_ϵ times.
5. Generate a random number bounded by n_{total}.
6. Find the layer and position to which the bit given by the random number belong.
7. Flip the corresponding bit to its opposite value.

A total of 25 experiments was performed for each bit error rate. The errors were injected only in the convolutional and fully connected layer weights. The range of BERs tested for each network in shown in Table 3.2 below:

Table 3.2: BER Range for different DNN

Network	**BER RANGE**
ResNet20	$10^{-6} - 10^{-5}$
ResNet32	$10^{-6} - 10^{-5}$
DenseNet	$5 \times 10^{-7} - 2.5 \times 10^{-6}$

The average accuracy of the 25 experiments at each BER was calculated. Figures 3.1,3.2,3.3 show the trends in classification accuracy on the 10,000 validation images of the CIFAR-10 at different BER levels.

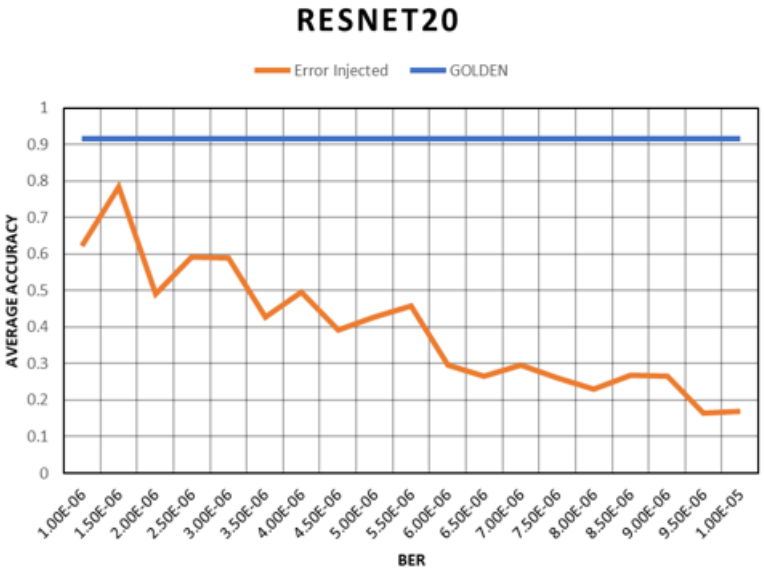

Figure 3.1: ResNet20 - Average Accuracy vs BER

It is observed that the accuracy degrades very rapidly from the golden

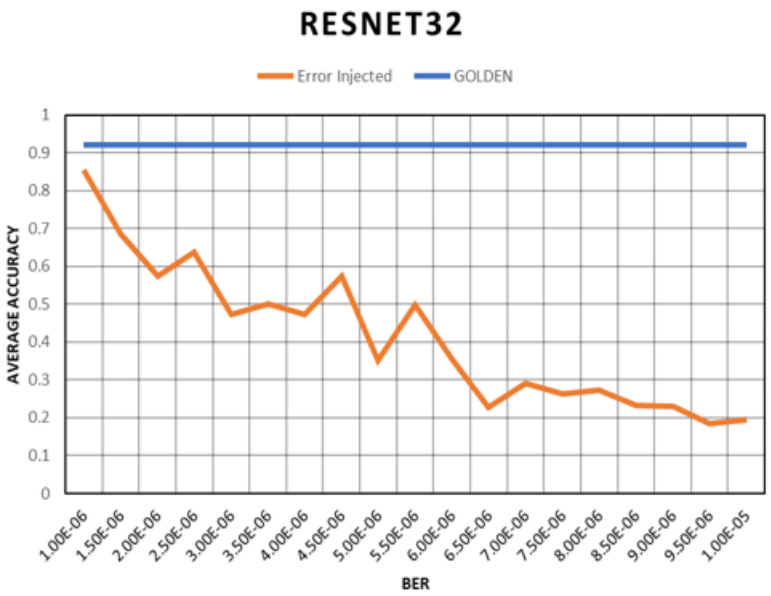

Figure 3.2: ResNet32 - Average Accuracy vs BER

model even at lower BERs. For example, DenseNet has approximately 0.75M parameters and with a BER of 8.5 x 10-7 the accuracy almost drops to about 55%. Note that this classification problem has a trivial number of 10 classes. However, these DNNs especially the DenseNet has about 100 convolution layers and is over-provisioned for this task. These networks might not seem fault tolerant because of the rapid degradation in the accuracy with BER but a closer look at the potential reasons is required. This model is based on the single-precision floating point format and even a single bit flip could change the weight by orders of magnitude if the exponent bits flip. Besides, with 40 to 50 such bit flips, the entire network could be corrupted. Strangely it was observed that with even less than ten bit-flips ResNet 20 has a worst-case

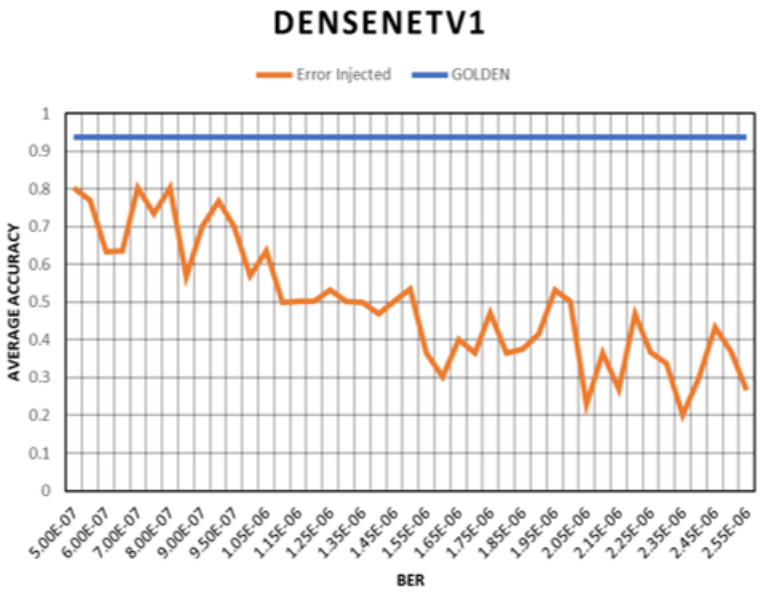

Figure 3.3: DenseNet - Average Accuracy vs BER

accuracy of 0.0678. This essentially means that even very few critical bit flips can completely corrupt the network's output.Hence, to find the most sensitive layer in ResNet 20, the weight distribution of each layer in the network is shown in Figures 3.4,3.5,3.6.

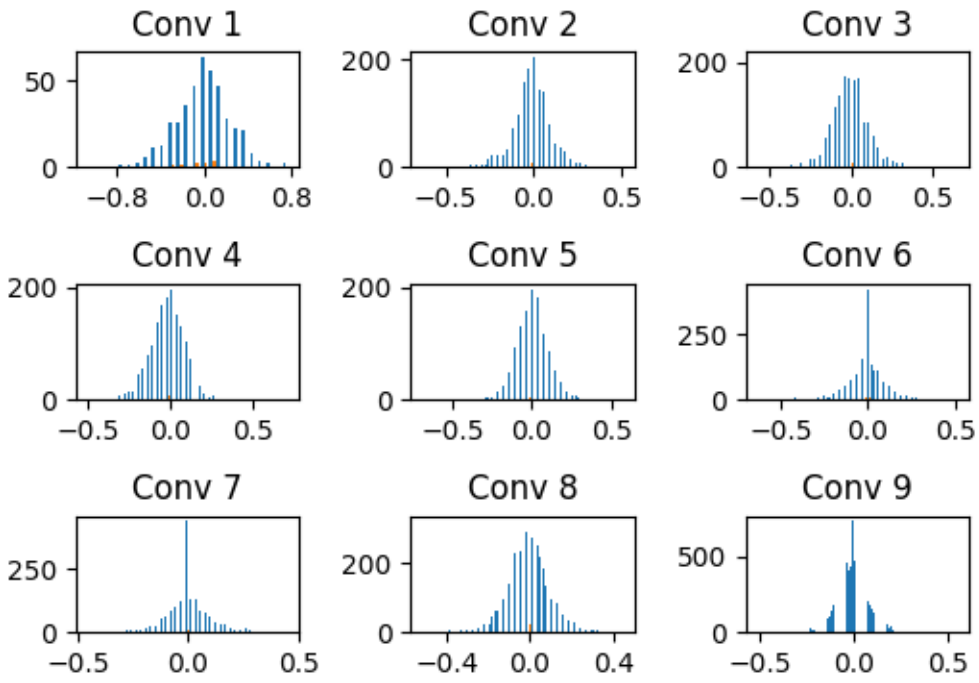

Figure 3.4: ResNet20: Weight Distribution of Convolutional Layers 1-9

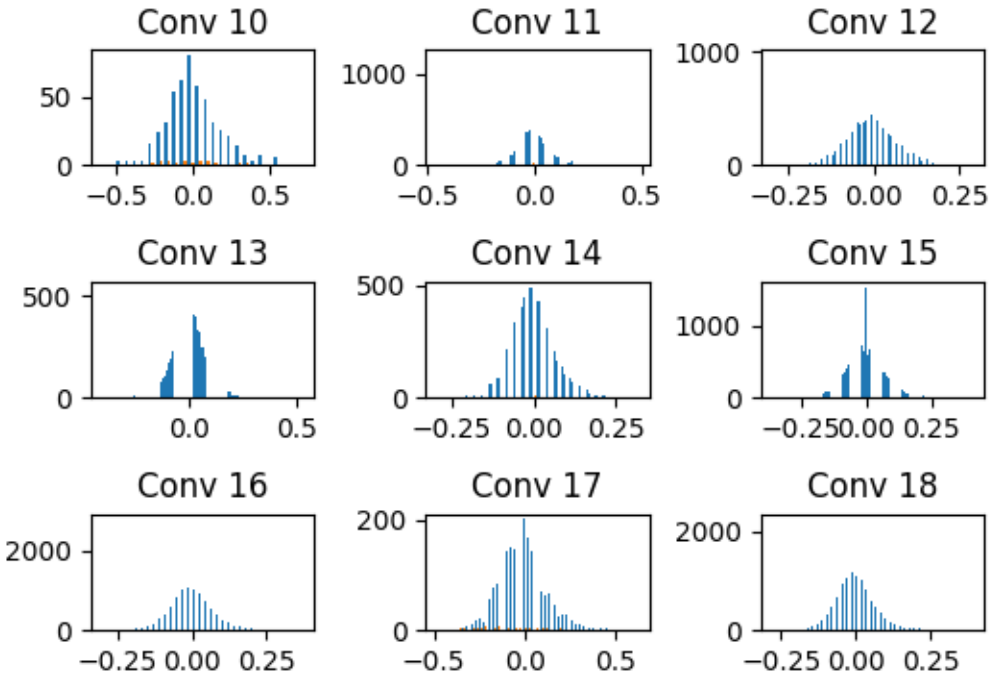

Figure 3.5: ResNet20: Weight Distribution of Convolutional Layers 10-18

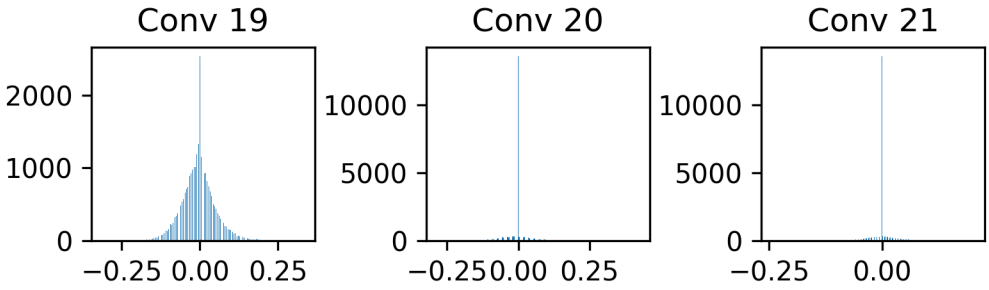

Figure 3.6: ResNet20: Weight Distribution of Convolutional Layers 19-21

The weight distribution of different layers show that the deeper we get into the network, the weights start getting close to 0. For very small weights, precision is important and a bit-flip in the exponent range could critically affect the outputs of the layers. Clearly, the last convolutional layer whose weights are very small and has a mode around 0 could be very sensitive to even a single bit flip. To verify this claim, 1 to 10 bit-errors were randomly injected into this layer and 25 experiments were carried out in each case. Figure 3.7 shows the worst-case classification accuracy for this range of bit errors.

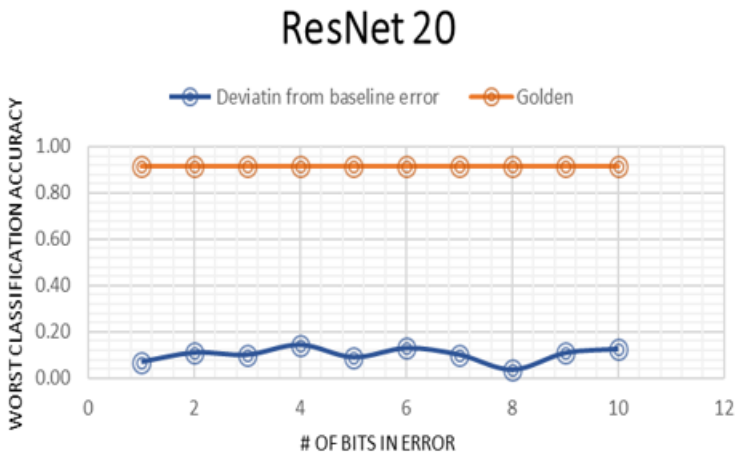

Figure 3.7: ResNet20 - Worst-case Accuracy vs # of bits in error

This trend shows that even a single critical bit error in the last convolutional layer could drop the overall accuracy to less than 10%. Evidently, the last convolutional layer is very sensitive to bit errors. To investigate the effects of bit error on the SoftMax output probabilities, two cases were selected: Golden Trained model and 10 random bit errors with a classification accuracy

on par with the golden model. A metric delta is defined as the difference between the top-1 and 2 SoftMax output probabilities for the cases correctly classified. A scatter plot of the delta values for the validation images is shown in Figure 3.8. Clearly, the network seems to work on par with 10 bit-errors with no variation of the SoftMax probabilities. Furthermore, this proves the claim that critical bits in error can cause catastrophic effects on the accuracy while non-critical bits have very less impact on the system.

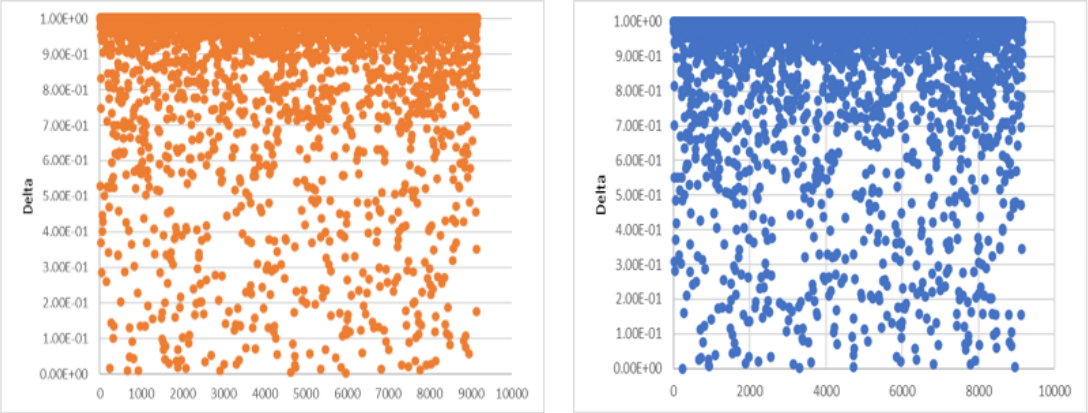

Figure 3.8: Distribution of Delta between top softmax probabilities (Left: Golden Trained Model, Right: Error Injected Model)

Chapter 4

Active Fault Tolerance in DNN Accelerators

It was observed that deep layers in the network have a very small dynamic range (close to 0). A 32-bit floating-point has several unused representations in the higher ranges. Hence, it would be interesting to see the behavior of the networks to bit errors when operated with different data types. To mitigate the error propagation and to improve the reliability of the DNN system different approaches have been proposed [10] [12]. This chapter will present a brief survey of these active fault-tolerant techniques for DNN systems.

4.1 Symptom-Based Error Detector (SED)

This type of detector utilizes application-specific symptoms under faults to detect anomalies. G. Li et al observed that the outputs of the activation

The entire chapter is a survey of fault tolerant techniques based on "Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Association for Computing Machinery" and "Christoph Schorn, Andre Guntoro, and Gerd Ascheid.Efficient on-line error detection and mitigation for deep neural network accelerators.In Barbara Gallina, Amund Skavhaug, and Friedemann Bitsch, editors,Computer Safety, Reliability, and Security, pages 205–219. Springer International Publishing, 2018".

layers right after the convolution layer were more influenced by bit errors: If an error makes the activation outputs very large, it likely leads to failures, and if it does not, it is likely to be benign. SEDs have two phases: learning and deployment. During the learning stage, with the golden trained model and with the representative test inputs, the range of fault free outputs at each layer [-X,Y] is calculated. For flexibility, an additional 10% cushion is added to this range [-1.1*X, 1.1*Y]. Once the detectors are derived, they are checked by the host which off-loads tasks to the DNN accelerator. At the end of execution of each layer, the outputs are usually stored in a global buffer [1] when the next layer's execution happens in the processing elements (PE). The detector is executed asynchronously by the host during this time by checking the values against the cushioned range. Experimental evaluations [10] show that the FIT rates of Eyeriss accelerator were reduced by 96% using the SED.

4.2 Selective Latch Hardening (SLH)

Latch hardening is a hardware error mitigation technique that adds redundant circuitry to latches to make them less sensitive to errors. Random bit flips in different positions can have different sensitivity. In a typical floating-point format, flips in the exponent bit might have a huge impact. Hence, selective hardening can be applied to certain critical bit latches on the data path. Different hardening techniques like Strike Suppression, redundant node and TMR can be applied to find the suitable one with more error and less cost. In fact, it was shown by Li et al [10] that a hybrid combination of all three

techniques reduced the FIT rate in Eyeriss by 100x while incurring 20% to 25% latch area overhead. The traditional ECC protection schemes might be a better solution for large memory arrays, as this technique could contribute a significant area overhead.

4.3 Weight Shifting

A set of predefined probing vectors can be applied to the network to detect the faulty weights [12]. As soon as a link or neuron is detected to be faulty [7], their weights are shifted to other fault free links of the neuron. In the case where an entire neuron is faulty, all the links are considered to be faulty. This is a self-recovery mechanism and does not require re-training or hardware repair. However, this method was evaluated only for single output two-layered neural networks and the scalability of this technique needs to be investigated.

4.4 Error Detection and Mitigation Network (EDMN)

Given a DNN model and the input image, a feature activation trace is generated by concatenating the feature activations of all the layers. This method is based on the idea that a bit-flip that causes anomaly will result in an anomaly in the feature activation trace. An Error Detection and Mitigation Network is built [11] whose input is the feature activation trace. It consists of two fully connected layers as shown in Figure 4.1. There are 2 output layers, one which has a single neuron that detects the error and the other output

layer has ‘n’ output layer neurons for an image classification problem with ‘n’ classes.

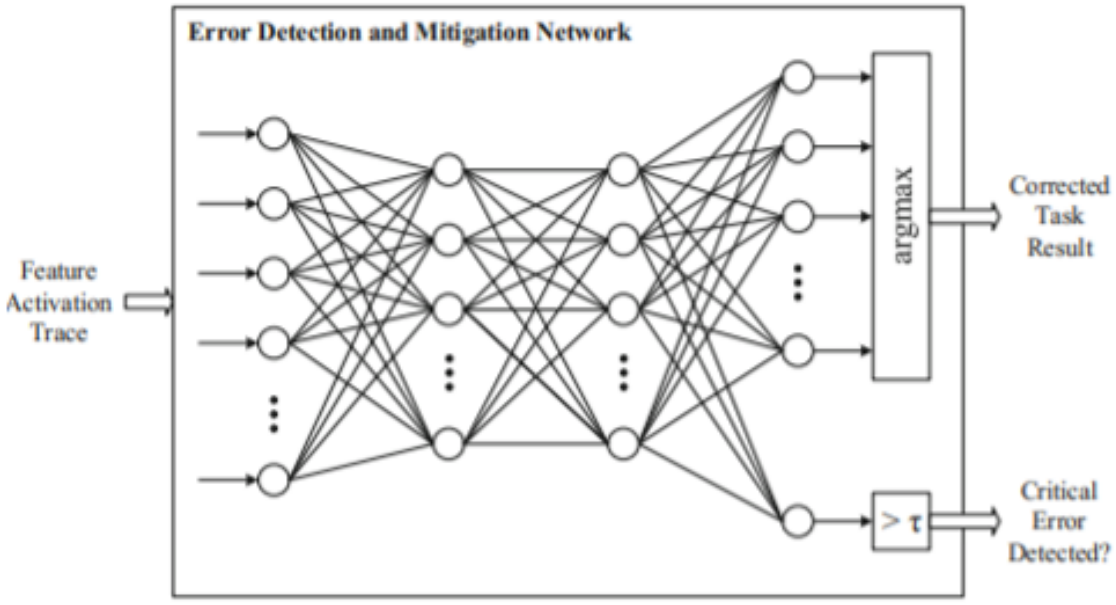

Figure 4.1: Error Detection and Mitigation Network (EDMN) [11]

The EDMN detects the anomaly and gives the corrected output class. EDMN is trained using three types of data:

1. Fault-free feature activations.
2. Feature activations with non-critical faults.
3. Feature activation with critical faults.

Experimental results [11] show that 96.16% of the critical errors were detected with a MAC computation overhead of just 0.41%.

Chapter 5

Conclusion

The bit error tolerance of three different DNN was studied by fault injection experiments. While these DNN are over-provisioned for a simple classification problem with 10 classes, critical bit flips tend to degrade the performance rapidly. It was observed that the weights of the deeper layers are more sensitive to bit flips because of the very small dynamic range close to 0. Moreover, this high sensitivity was observed due to the 32-bit floating-point format. A similar set of fault injection experiments on these DNN for other data types including half-precision floating-point and integer data types could bring valuable insights on the fault sensitivity of the networks. Furthermore, to tackle the performance degradation active error tolerance procedures need to be adapted. While there is an area overhead with additional redundancy or error detection mechanisms, the performance improvement is very promising, especially for EDMN methodology.

Bibliography

[1] Y. Chen, T. Krishna, J. Emer, and V. Sze. 14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 262–263, 2016.

[2] Y. Chen, T. Yang, J. Emer, and V. Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[3] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. Shidiannao: Shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 92–104, 2015.

[4] El Mahdi El Mhamdi and Rachid Guerraoui. When neurons fail - technical report. page 19, 2016.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[7] C. Khunasaraphan, K. Vanapipat, and C. Lursinsap. Weight shifting techniques for self-recovery neural networks. *IEEE Transactions on Neural Networks*, 5(4):651–658, 1994.

[8] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[10] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Association for Computing Machinery, 2017.

[11] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. Efficient online error detection and mitigation for deep neural network accelerators. In Barbara Gallina, Amund Skavhaug, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 205–219. Springer International Publishing, 2018.

[12] T. Tanprasert, C. Tanprasert, and C. Lursinsap. Probing technique for neural net fault detection. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 2, pages 1001–1005 vol.2, 1996.

[13] C. Torres-Huitzil and B. Girau. Fault and error tolerance in neural networks: A review. *IEEE Access*, 5:17322–17341, 2017.

[14] G. Zhou, J. Zhou, and H. Lin. Research on nvidia deep learning accelerator. In *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 192–195, 2018.